Homework 3

Policy Gradients

I would like to use one of my extra-late days for this assignment, as I wanted to do the bonus part.

Experiment 1

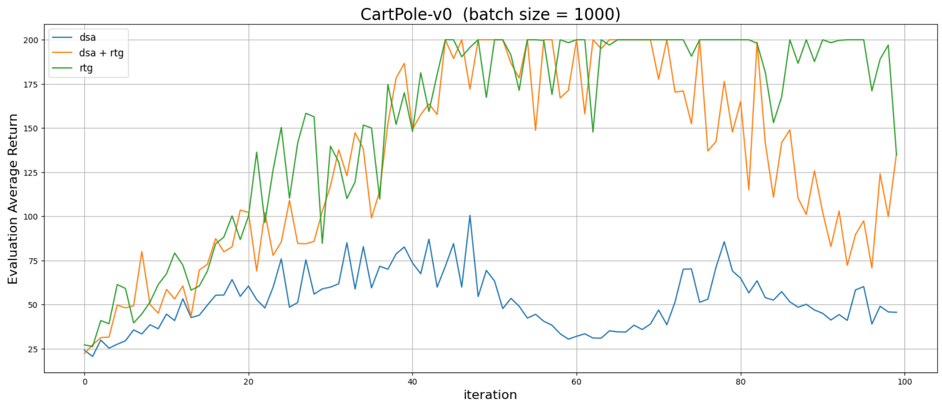

Figure 1: Effect of different hyper-parameters usage (i.e. reward to go and stanstandardization of advantages) on the evaluation average return on the cart pole. The batch size was fixed to 1000 with a number of iteration of 100.

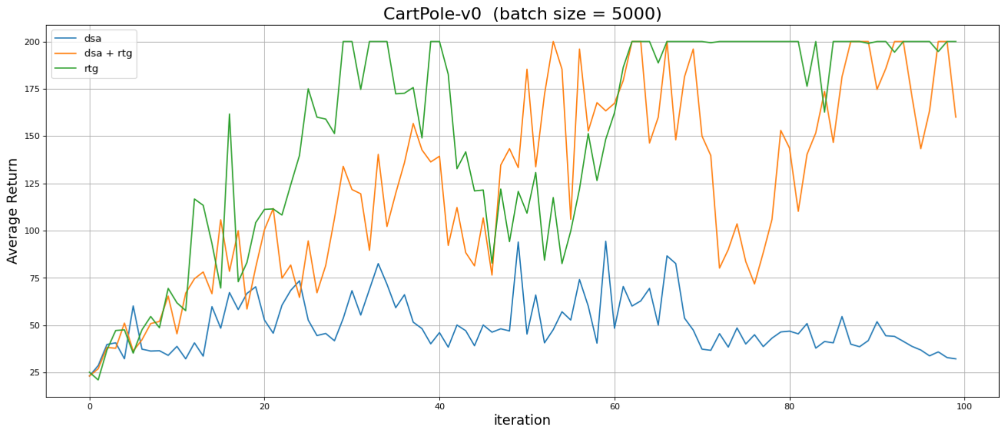

Figure 2: Effect of different hyper-parameters usage (i.e. reward to go and stanstandardization of advantages) on the evaluation average return on the cart pole. The batch size was fixed to 5000 with a number of iteration of 100.

Q: Which value estimator has better performance without advantage-standardization: the trajectory- centric one, or the one using reward-to-go?
For both Fig. 1 and Fig. 2, it can be noted that without the use of the reward to go, the policy does much less well than with.

Q: Did advantage standardization help?
Usage of advantage standardization help a bit as green curves are always a bit upper than the oranges one.It also seems to helping in the convergence of the algorithm.

Q: Did the batch size make an impact?
It seems to not making a huge difference. One may notice that experiment using only reward to-go Fig. 2 converge faster to the maximum amount of evaluation average return of 200 than in Fig. 1.

Here is the list of commands executed for this experiment:
make sure to use the default `config.yml` given in Section Appendix (updated default configuration file)

```
# command 1
python run_hw3.py env_name=CartPole-v0 n_iter=100 batch_size=1000 \
estimate_advantage_args.standardize_advantages=false \
exp_name=q1_sb_no_rtg_dsa rl_alg=reinforce

# command 2
python run_hw3.py env_name=CartPole-v0 n_iter=100 batch_size=1000 \
estimate_advantage_args.standardize_advantages=false \
estimate_advantage_args.reward_to_go=true exp_name=q1_sb_rtg_dsa \
rl_alg=reinforce

# command 3
python run_hw3.py env_name=CartPole-v0 n_iter=100 batch_size=1000\
estimate_advantage_args.reward_to_go=true exp_name=q1_sb_rtg_na\ rl_alg=reinforce

# command 4
python run_hw3.py env_name=CartPole-v0 n_iter=100 batch_size=5000 \
estimate_advantage_args.standardize_advantages=false \
exp_name=q1_lb_no_rtg_dsa rl_alg=reinforce

# command 5
python run_hw3.py env_name=CartPole-v0 n_iter=100 batch_size=5000 \
estimate_advantage_args.standardize_advantages=false \
estimate_advantage_args.reward_to_go=true exp_name=q1_lb_rtg_dsa \
rl_alg=reinforce

# command 6
python run_hw3.py env_name=CartPole-v0 n_iter=100 batch_size=5000 \
estimate_advantage_args.reward_to_go=true exp_name=q1_lb_rtg_na \
rl_alg=reinforce
```

Experiment 2

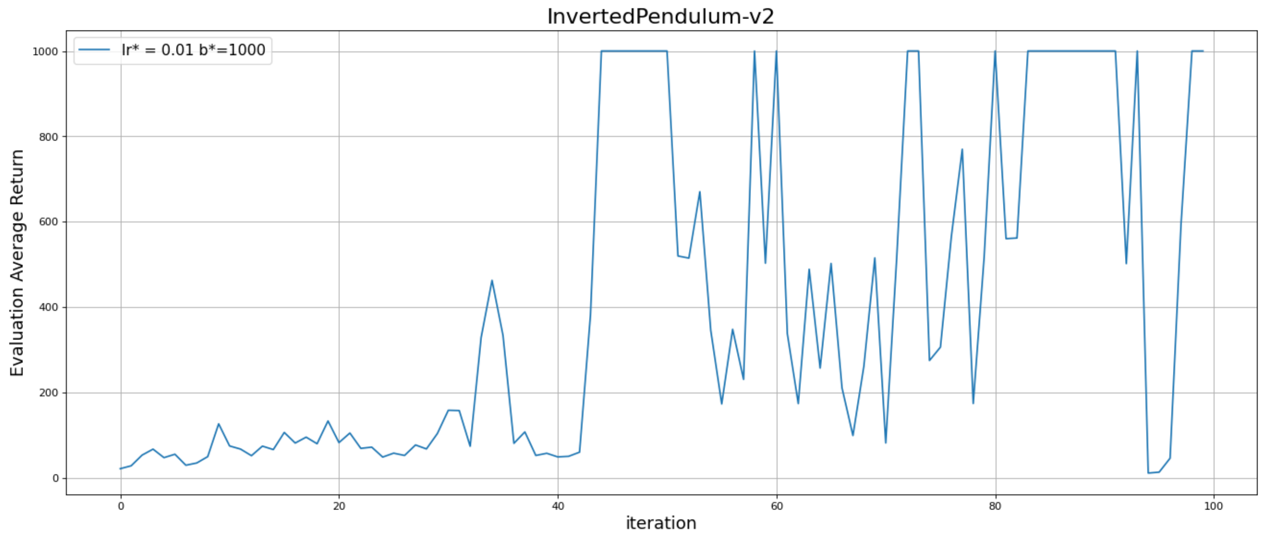

Figure 3: Evaluation average return of reinforce algorithm on the Inverted Pendulum task. The neural network has 2 layers consisting of 64 neurons, the episode length is set to 1000, the discount factor to 0.9, optimal batch size and learning rate value found are respectively 1000 and 0.01. Note that reward-to-go was used to run this experiment.

Here is the list of commands executed for this experiment:

```
# command 1
python run_hw3.py env_name=InvertedPendulum-v2 ep_len=1000 \
estimate_advantage_args.discount=0.9 n_iter=100 \
computation_graph_args.n_layers=2 computation_graph_args.size=64 \
batch_size=1000 computation_graph_args.learning_rate=0.01 \
estimate_advantage_args.reward_to_go=true exp_name=q2_b1000_r0.02 \
rl_alg=reinforce
```

Experiment 3

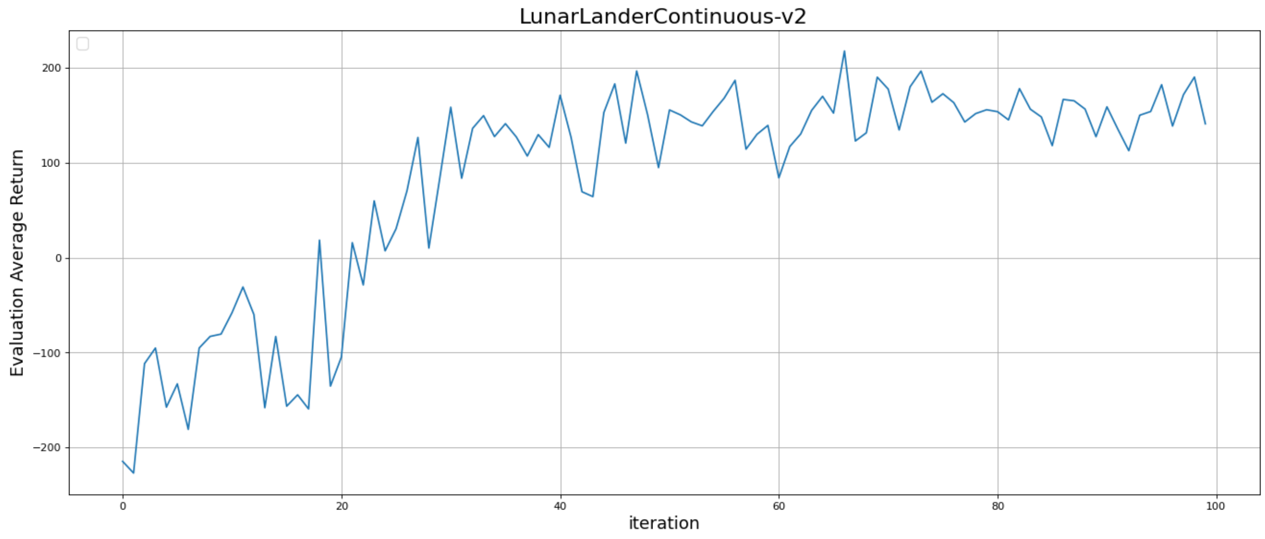

Figure 4: Evaluation average return of reinforce algorithm on the Lunar Lander Continuous task. The neural network has 2 layers consisting of 64 neurons, the episode length is set to 1000, the discount factor to 0.99, the learning rate was set to 0.005, and batch size was fixed to 40000. Note that I had to set the train batch size equals to the batch size to achieve the expected results. Reward-to-go and a neural network baseline were used to run this experiment.

Here is the list of commands executed for this experiment:

```
# command 1
python run_hw3.py env_name=LunarLanderContinuous-v2 ep_len=1000 \
estimate_advantage_args.discount=0.99 n_iter=100 \
computation_graph_args.n_layers=2 computation_graph_args.size=64 \
batch_size=40000 train_batch_size=40000 \
computation_graph_args.learning_rate=0.005 \
estimate_advantage_args.reward_to_go=true \
estimate_advantage_args.nn_baseline=true rl_alg=reinforce \
exp_name=q3_b40000_r0.005
```

Experiment 4

Experiment 4.1

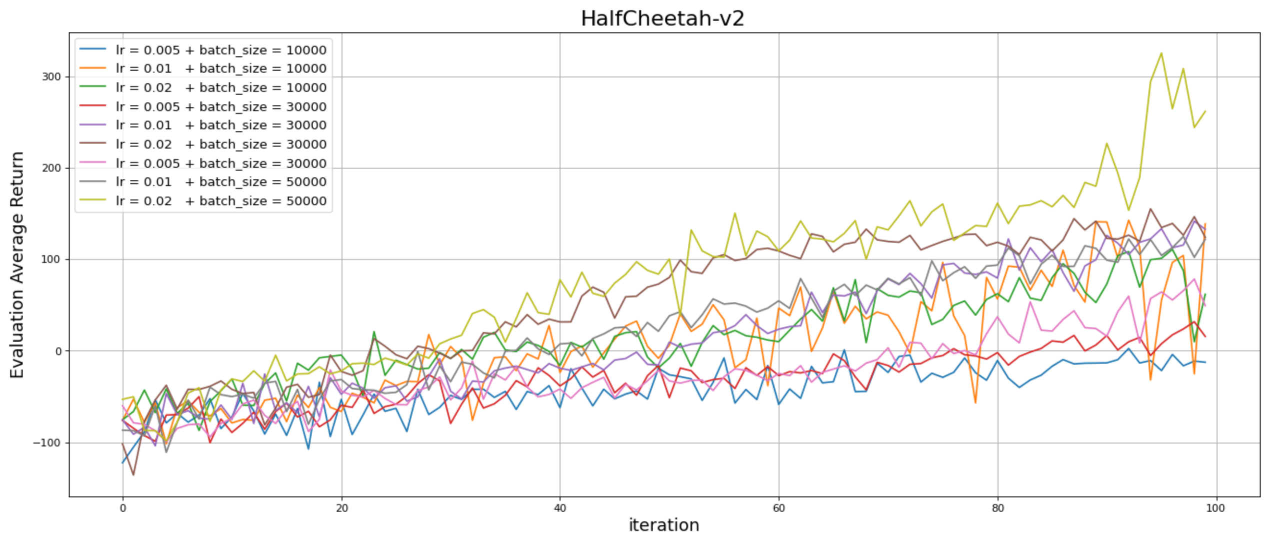

Figure 5: Effect of different batch size and learning rate on the evaluation average return of reinforce algorithm on the half cheetah task. The neural network has 2 layers consisting of 32 neurons, number of iteration was set to 100, the episode length is set to 150 for training purpose, the discount factor to 0.95. Note that I had to set the train batch size equals to the batch size to achieve the expected results. Reward-to-go and a neural network baseline were used to run this experiment. Clearly the best learning curve is one that uses a batch size of 50000 and a learning rate of 0.02. Some may notice that as soon as the batch size is either 50000 or 30000 the best learning curves are those that use a learning rate of 0.02. When the learning rate is 0.02 and the batch size is 10000 the learning is as good as for the other batch sizes at the beginning but stagnates as the iterations go on and is overtaken by the learning curves that have a learning rate of 0.01. For the curves with a learning rate of 0.05 the learning does not learn much and struggles to exceed an average evaluation return of 0.

Here is the list of commands executed for this experiment:

```
# command 1
python run_hw3.py env_name=HalfCheetah-v2 ep_len=150 \
estimate_advantage_args.discount=0.95 n_iter=100 \
computation_graph_args.n_layers=2 computation_graph_args.size=32 \
batch_size=10000 train_batch_size=10000 \
computation_graph_args.learning_rate=0.005 \
estimate_advantage_args.reward_to_go=true \
estimate_advantage_args.nn_baseline=true rl_alg=reinforce \
exp_name=q4_search_b10000_lr0.005_rtg_nnbaseline

# command 2
python run_hw3.py env_name=HalfCheetah-v2 ep_len=150 \
estimate_advantage_args.discount=0.95 n_iter=100 \
computation_graph_args.n_layers=2 computation_graph_args.size=32 \
batch_size=10000 train_batch_size=10000 \
computation_graph_args.learning_rate=0.01 \
estimate_advantage_args.reward_to_go=true \
estimate_advantage_args.nn_baseline=true rl_alg=reinforce \
exp_name=q4_search_b10000_lr0.01_rtg_nnbaseline

# command 3
python run_hw3.py env_name=HalfCheetah-v2 ep_len=150 \
estimate_advantage_args.discount=0.95 n_iter=100 \
computation_graph_args.n_layers=2 computation_graph_args.size=32 \
batch_size=10000 train_batch_size=10000 \
computation_graph_args.learning_rate=0.02 \
estimate_advantage_args.reward_to_go=true \
estimate_advantage_args.nn_baseline=true rl_alg=reinforce \
exp_name=q4_search_b10000_lr0.02_rtg_nnbaseline

# command 4
python run_hw3.py env_name=HalfCheetah-v2 ep_len=150 \
estimate_advantage_args.discount=0.95 n_iter=100 \
computation_graph_args.n_layers=2 computation_graph_args.size=32 \
batch_size=30000 train_batch_size=30000 \
computation_graph_args.learning_rate=0.005 \
estimate_advantage_args.reward_to_go=true \
estimate_advantage_args.nn_baseline=true rl_alg=reinforce \
exp_name=q4_search_b30000_lr0.005_rtg_nnbaseline

# command 5
python run_hw3.py env_name=HalfCheetah-v2 ep_len=150 \
estimate_advantage_args.discount=0.95 n_iter=100 \
computation_graph_args.n_layers=2 computation_graph_args.size=32 \
batch_size=30000 train_batch_size=30000 \
computation_graph_args.learning_rate=0.01 \
estimate_advantage_args.reward_to_go=true \
estimate_advantage_args.nn_baseline=true rl_alg=reinforce \
exp_name=q4_search_b30000_lr0.01_rtg_nnbaseline
```

```
# command 6
python run_hw3.py env_name=HalfCheetah-v2 ep_len=150 \
estimate_advantage_args.discount=0.95 n_iter=100 \
computation_graph_args.n_layers=2 computation_graph_args.size=32 \
batch_size=30000 train_batch_size=30000 \
computation_graph_args.learning_rate=0.02 \
estimate_advantage_args.reward_to_go=true \
estimate_advantage_args.nn_baseline=true rl_alg=reinforce \
exp_name=q4_search_b30000_lr0.02_rtg_nnbaseline

# command 7
python run_hw3.py env_name=HalfCheetah-v2 ep_len=150 \
estimate_advantage_args.discount=0.95 n_iter=100 \
computation_graph_args.n_layers=2 computation_graph_args.size=32 \
batch_size=50000 train_batch_size=50000 \
computation_graph_args.learning_rate=0.005 \
estimate_advantage_args.reward_to_go=true \
estimate_advantage_args.nn_baseline=true rl_alg=reinforce \
exp_name=q4_search_b50000_lr0.005_rtg_nnbaseline

# command 8
python run_hw3.py env_name=HalfCheetah-v2 ep_len=150 \
estimate_advantage_args.discount=0.95 n_iter=100 \
computation_graph_args.n_layers=2 computation_graph_args.size=32 \
batch_size=50000 train_batch_size=50000 \
computation_graph_args.learning_rate=0.01 \
estimate_advantage_args.reward_to_go=true \
estimate_advantage_args.nn_baseline=true rl_alg=reinforce \
exp_name=q4_search_b50000_lr0.01_rtg_nnbaseline

# command 9
python run_hw3.py env_name=HalfCheetah-v2 ep_len=150 \
estimate_advantage_args.discount=0.95 n_iter=100 \
computation_graph_args.n_layers=2 computation_graph_args.size=32 \
batch_size=50000 train_batch_size=50000 \
computation_graph_args.learning_rate=0.02 \
estimate_advantage_args.reward_to_go=true \
estimate_advantage_args.nn_baseline=true rl_alg=reinforce \
exp_name=q4_search_b50000_lr0.02_rtg_nnbaseline
```

Experiment 4.2

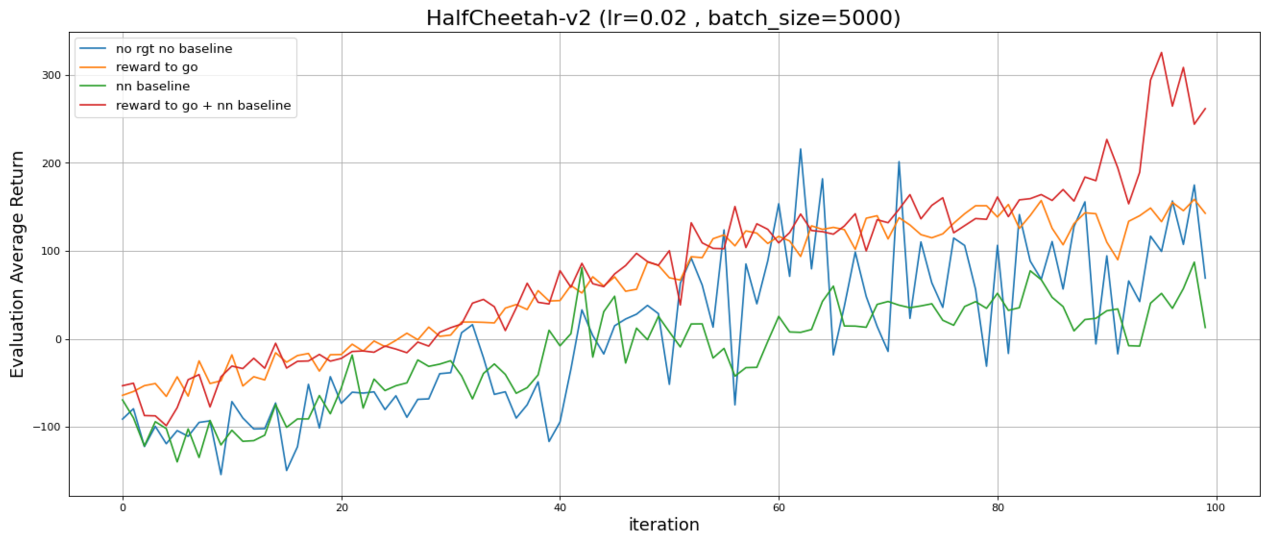

Figure 6: Effect of reward to go and neural network baseline the evaluation average return of reinforce algorithm on the half cheetah task. The neural network has 2 layers consisting of 32 neurons, number of iteration was set to 100, the discount factor to 0.99 and the learning rate was set to 0.001. Note that in addition to GAE both reward-to-go and neural network baseline were used to run the experiments. GAE allows changing the trade-off between bias and variance during the training. GAE's lambda set to 0 will result in a low variance but high biased advantage estimate, which leads to poor results as shown by the blue curve. GAE's lambda of 1 will result in a low biased but high variance advantage estimate, which leads to better result than when lambda was set to 0. Best result is when the lambda is set to 0.95 as show by the orange curve but 0.99 gives reasonable results.

Here is the list of commands executed for this experiment:

```
# command 1
python run_hw3.py env_name=HalfCheetah-v2 ep_len=150 rl_alg=reinforce \
estimate_advantage_args.discount=0.95 n_iter=100 computation_graph_args.n_layers=2 \
computation_graph_args.size=32 batch_size=50000 train_batch_size=50000 \
computation_graph_args learning_rate=0.02 exp_name=q4_b50000_r0.02

# command 2
python run_hw3.py env_name=HalfCheetah-v2 ep_len=150 rl_alg=reinforce \
estimate_advantage_args.discount=0.95 n_iter=100 computation_graph_args.n_layers=2 \
computation_graph_args.size=32 batch_size=50000 train_batch_size=50000 \
computation_graph_args.learning_rate=0.02 estimate_advantage_args.reward_to_go=true \
exp_name=q4_b50000_r0.02_rtg

# command 3
python run_hw3.py env_name=HalfCheetah-v2 ep_len=150 rl_alg=reinforce \
estimate_advantage_args.discount=0.95 n_iter=100 computation_graph_args.n_layers=2 \
computation_graph_args.size=32 batch_size=50000 train_batch_size=50000 \
computation_graph_args.learning_rate=0.02 estimate_advantage_args.nn_baseline=true \
exp_name=q4_b50000_r0.02_nnbaseline

# command 4
python run_hw3.py env_name=HalfCheetah-v2 ep_len=150 rl_alg=reinforce \
estimate_advantage_args.discount=0.95 n_iter=100 computation_graph_args.n_layers=2 \
computation_graph_args.size=32 batch_size=50000 train_batch_size=50000 \
computation_graph_args.learning_rate=0.02 estimate_advantage_args.reward_to_go=true \
estimate_advantage_args.nn_baseline=true exp_name=q4_b50000_r0.02_rtg_nnbaseline
```

Experiment 5

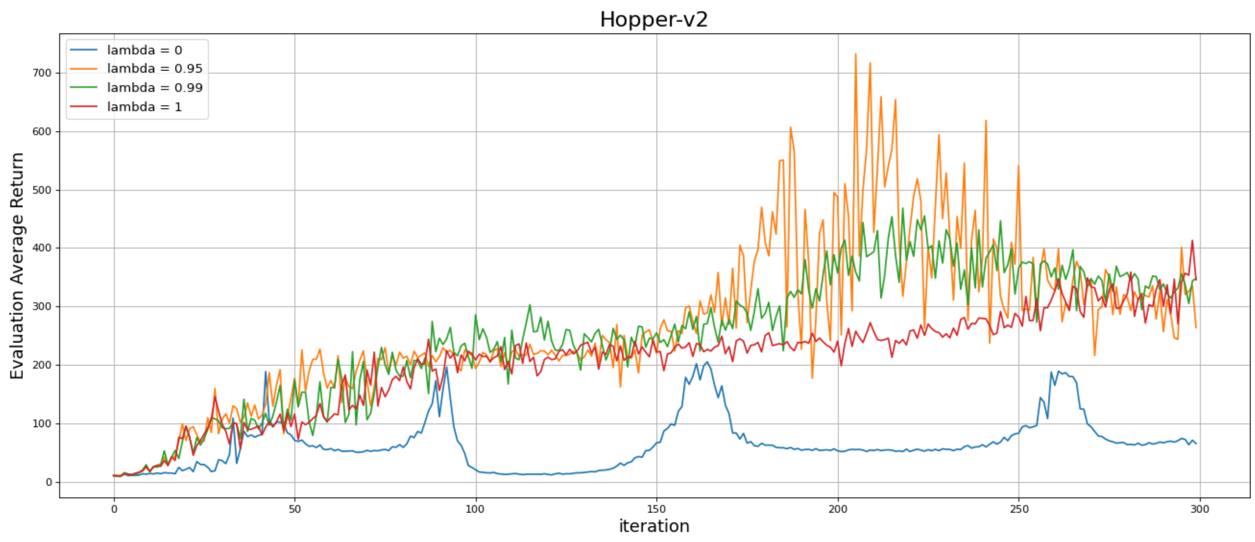

Figure 7: Effect of Generalized Advantage Estimation's lambda (GAE) coefficient on the evaluation average return of reinforce algorithm on the hopper task. The neural network has 2 layers consisting of 32 neurons, number of iteration was set to 300, the episode length is set to 150 for training purpose, the discount factor to 0.95. Note that I had to set the train batch size equals to the batch size to achieve the expected results. The optimal batch size and learning from the previous experiment shown in Fig.5 were used for this one (i.e batch size of 50000 and learning rate set to 0.02). Clearly, we can see that the reward-to-go is helping a lot in learning. Coupled with a neural network baseline, this gives better performance but not as notifiable as when reward-to-go is used or not

Here is the list of commands executed for this experiment:

```
# command 1
python run_hw3.py env_name=Hopper-v2 ep_len=1000 rl_alg=reinforce \
estimate_advantage_args.discount=0.99 n_iter=300 computation_graph_args.n_layers=2 \
computation_graph_args.size=32 batch_size=2000 computation_graph_args.learning_rate=0.001 \
estimate_advantage_args.reward_to_go=true estimate_advantage_args.nn_baseline=true \
estimate_advantage_args.gae_lambda=0 exp_name=q5_b2000_r0.001_lambda0

# command 2
python run_hw3.py env_name=Hopper-v2 ep_len=1000 rl_alg=reinforce \
estimate_advantage_args.discount=0.99 n_iter=300 computation_graph_args.n_layers=2 \
computation_graph_args.size=32 batch_size=2000 computation_graph_args.learning_rate=0.001 \
estimate_advantage_args.reward_to_go=true estimate_advantage_args.nn_baseline=true \
estimate_advantage_args.gae_lambda=0.95 exp_name=q5_b2000_r0.001_lambda0.95

# command 3
python run_hw3.py env_name=Hopper-v2 ep_len=1000 rl_alg=reinforce \
estimate_advantage_args.discount=0.99 n_iter=300 computation_graph_args.n_layers=2 \
computation_graph_args.size=32 batch_size=2000 computation_graph_args.learning_rate=0.001 \
estimate_advantage_args.reward_to_go=true estimate_advantage_args.nn_baseline=true \
estimate_advantage_args.gae_lambda=0.99exp_name=q5_b2000_r0.001_lambda0.99

# command 4
python run_hw3.py env_name=Hopper-v2 ep_len=1000 rl_alg=reinforce \
estimate_advantage_args.discount=0.99 n_iter=300 computation_graph_args.n_layers=2 \
computation_graph_args.size=32 batch_size=2000 computation_graph_args.learning_rate=0.001 \
estimate_advantage_args.reward_to_go=true estimate_advantage_args.nn_baseline=true \
estimate_advantage_args.gae_lambda=1 exp_name=q5_b2000_r0.001_lambda1
```

Experiment 6

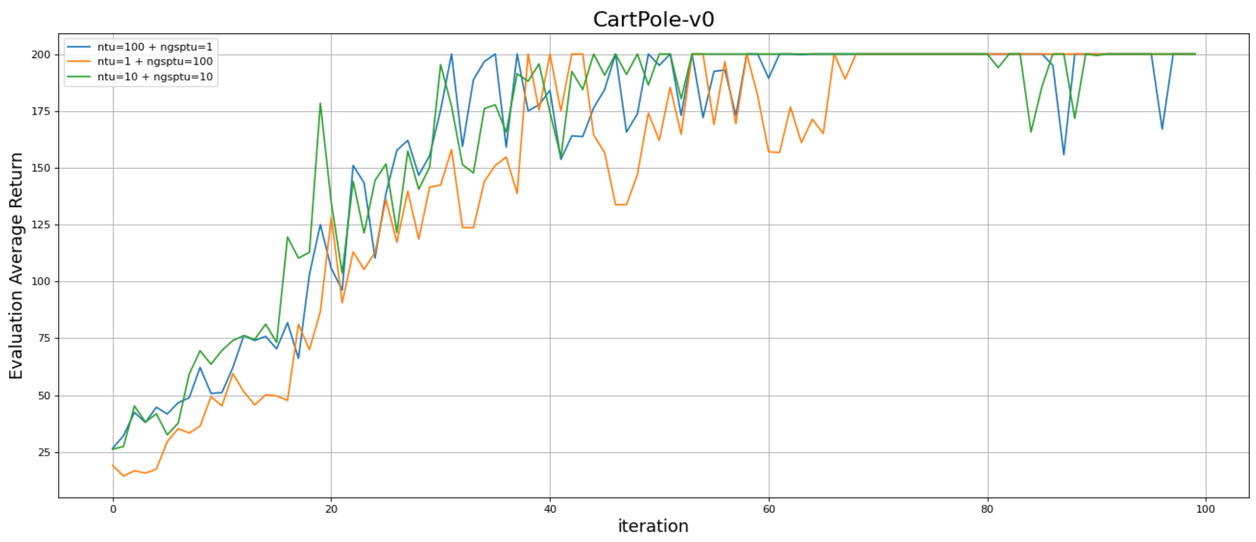

Figure 8: Effect of number of target updates (ntu) and number of gradient steps per target update (ngsptu) of actor-critic algorithm on the Cart Pole task. Number of iteration was set to 1000. Note that I had to set the train batch size equals to the batch size to achieve the expected results. We can notice that both ntu and ngsptu are both important to achieved good result on cart pole. The best learning curve is the one with ntu set to 10 and ngsptu set to 10 which suggest that optimal values are a trade-off between the two values as fixed one of them to 100 does not lead to the best performance.

Here is the list of commands executed for this experiment:

```
# command 1
python run_hw3.py env_name=CartPole-v0 n_iter=100 rl_alg=ac \
batch_size=1000 train_batch_size=1000 exp_name=q6_100_1 \
computation_graph_args.num_target_updates=100 \
computation_graph_args.num_grad_steps_per_target_update=1

# command 2
python run_hw3.py env_name=CartPole-v0 n_iter=100 rl_alg=ac \
batch_size=1000 train_batch_size=1000 exp_name=q6_1_100 \
computation_graph_args.num_target_updates=1 \
computation_graph_args.num_grad_steps_per_target_update=100

# command 3
python run_hw3.py env_name=CartPole-v0 n_iter=100 rl_alg=ac \
batch_size=1000 train_batch_size=1000 exp_name=q6_10_10 \
computation_graph_args.num_target_updates=10 \
computation_graph_args.num_grad_steps_per_target_update=10
```

Experiment 7

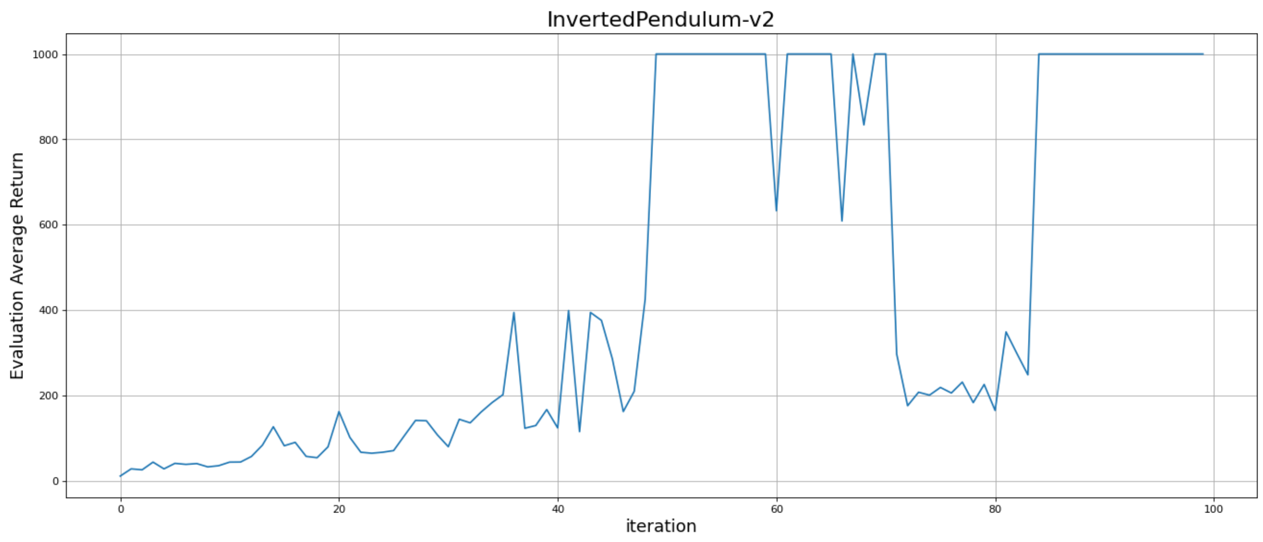

Figure 9: Evaluation average return of actor-critic algorithm on the Inverted Pendulum task. The neural network has 2 layers consisting of 64 neurons, number of iteration was set to 100. the episode length is set to 1000, the discount factor to 0.95, the learning rate was set to 0.01, and batch size was fixed to 5000. Number of target updates was set to 10 and number of gradient steps per target update was also set to 10.

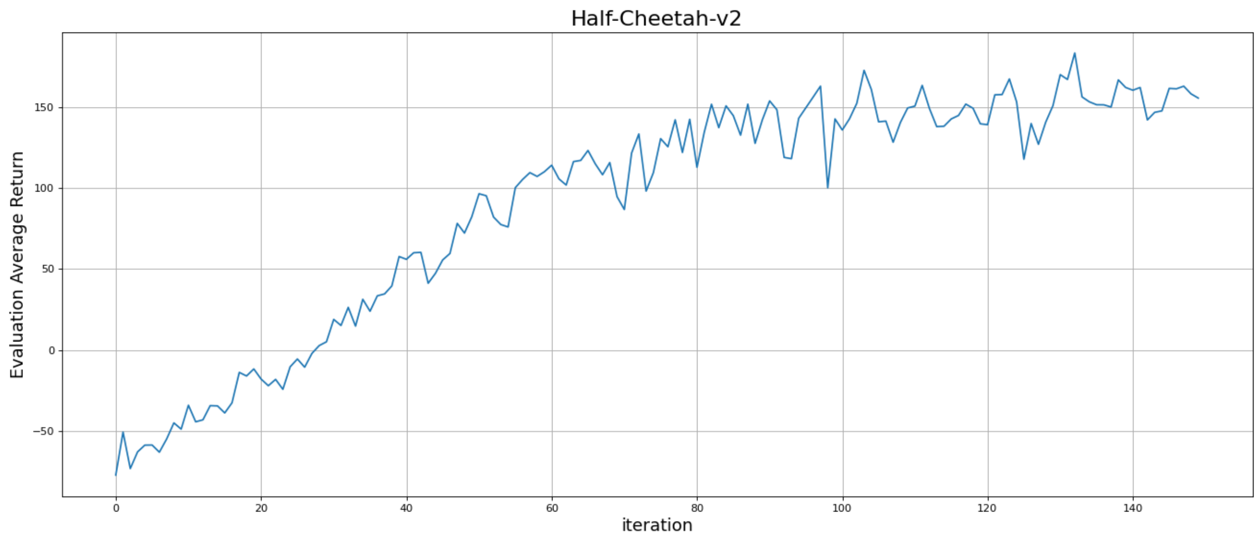

Figure 10: Evaluation average return of actor-critic algorithm on the Inverted Pendulum task. The neural network has 2 layers consisting of 32 neurons, number of iteration was set to 150, the episode length is set to 150, the discount factor to 0.90, the learning rate was set to 0.02, and batch size was fixed to, 30000 and evaluation batch size to 1500. Number of target updates was set to 10 and number of gradient steps per target update was also set to 10. Not that to achieve the expected result, I had to set the train batch size equals to the batch size.

Here is the list of commands executed for this experiment:

```
# command 1
python run_hw3.py env_name=InvertedPendulum-v2 rl_alg=ac ep_len=1000\
estimate_advantage_args.discount=0.95 n_iter=100\
computation_graph_args.n_layers=2 computation_graph_args.size=64\
batch_size=5000 computation_graph_args.learning_rate=0.01\
exp_name=q7_10_10\
computation_graph_args.num_target_updates=10\
computation_graph_args.num_grad_steps_per_target_update=10

# command 2
python run_hw3.py env_name=HalfCheetah-v2 rl_alg=ac ep_len=150\
estimate_advantage_args.discount=0.90 \
scalar_log_freq=1 n_iter=150 computation_graph_args.n_layers=2 \
computation_graph_args.size=32 batch_size=30000 \
train_batch_size=30000 eval_batch_size=1500 \
computation_graph_args.learning_rate=0.02 exp_name=q7_c_10_10 \
computation_graph_args.num_target_updates=10 \
computation_graph_args.num_grad_steps_per_target_update=10
```

Experiment 8

Hint: Should the critic be trained with this generated data? Try with and without and include your findings in the report.

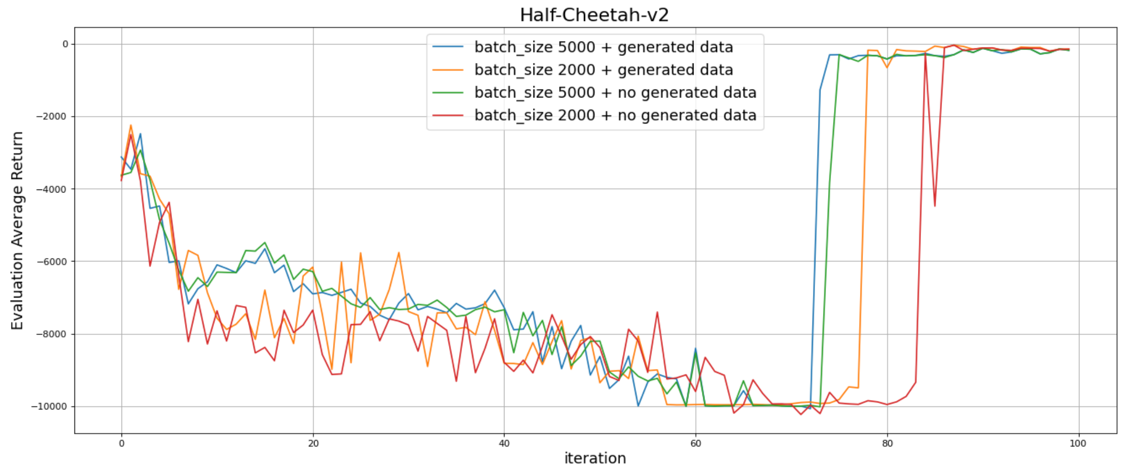

Figure 11: Evaluation average return of Dyna agent on half-cheetah task. The neural network has 2 layers consisting of 32 neurons, number of iteration was set to 100,the learning rate was set to 0.01, batch size was fixed to 5000 or to 2000 and training batch size to 1024. Some critic were trained using only the data sampled from the environment and some were using both data from the environment and world's model generated ones. As showing by the curves for the 70-ish first iterations, dyna is not very good to solving the task, probably because the agent is learning a good world model as suggested by the end of the learning curve where the agent is able to achieved nears 0 evaluation average return. Some may notify that using both data from the environment and world's model generated ones is leading to a solution faster, especially when the batch size is set to 2000.

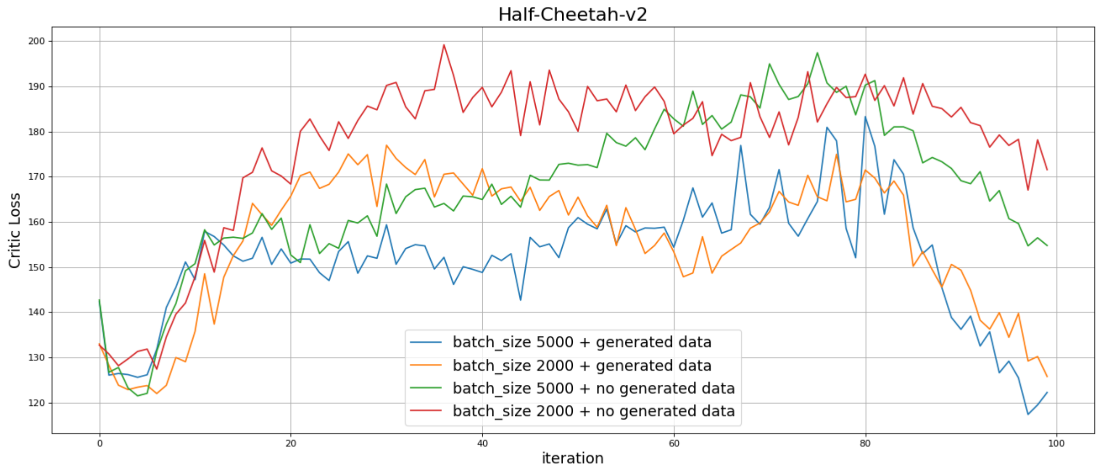

Figure 12: Critic loss of the above experiment. Critic loss is increasing during the learning of the world's model. After that, loss is reducing, especially when the dyna agent is training its critic using both data from the environment and world's model generated ones. This can be seen as the fact that training dyna's critic with generated data leads to more sample efficiency.

Here is the list of commands executed for this experiment:

rem: actor-critic curves are extracted from the results of the experiment 7.

```
# command 1
python run_hw3.py exp_name=q8_cheetah_n500_arch1x32 env_name=cheetah-ift6163-v0 \
estimate_advantage_args.discount=0.95 computation_graph_args.n_layers=2 \
computation_graph_args.size=32 \
computation_graph_args.learning_rate=0.01 scalar_log_freq=1 n_iter=100 \
batch_size=5000 \
train_batch_size=1024 rl_alg=dyna

# command 2
python run_hw3.py exp_name=q8_cheetah_n500_arch1x32_2 env_name=cheetah-ift6163-v0  \
estimate_advantage_args.discount=0.95 computation_graph_args.n_layers=2 \
computation_graph_args.size=32 \
computation_graph_args.learning_rate=0.01 scalar_log_freq=1 n_iter=100 \
batch_size=2000 \
train_batch_size=1024 rl_alg=dyna
```

Experiment BONUS: PPO Clipping

The objective used so far by the actor-critic was the policy gradient's objective:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta \left(a_t \mid s_t\right) \hat{A}_t\right]$$

One of the bonus consisted of altering that objective by the clipped objective from PPO:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \operatorname{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right) \hat{A}_t\right)\right]$$

where $r_t(\theta)$ is the ratio between current policy and an old policy copy.

I decided to compare the two losses on the difficult tasks of the critical actor in experiment 7 and reported the following results:

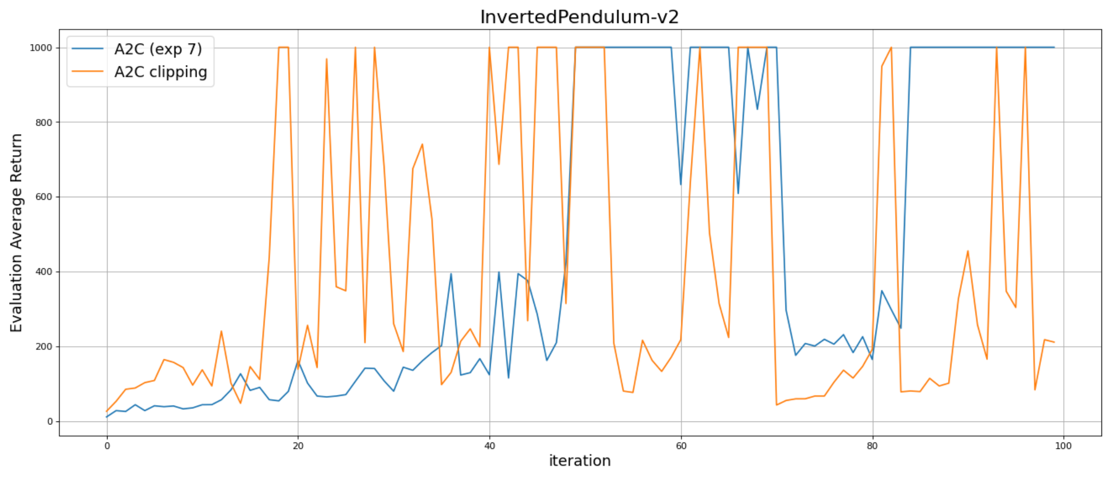

Figure 13: Performance of actor-critic algorithm using PPO clipping or not on the Inverted Pendulum task. The configuration for that experiment is the as the one used in the experiment 7. We can see that, as expected, the actor-critic using the PPO clipping term is converging much faster to the solution than the one using classical policy gradient objective.

rem: See the next page for the other experiment.

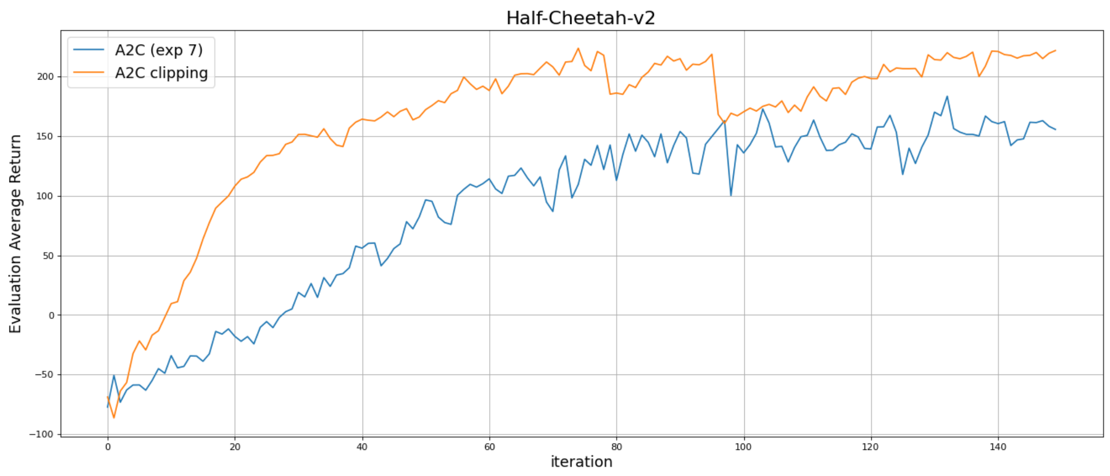

Figure 14: Performance of actor-critic algorithm using PPO clipping or not on the Half-Cheetath task. The configuration for that experiment is the as the one used in the experiment 7. Here we can notice that in addition to converging much faster to the solution ($\simeq 150$ at iteration 30 for A2C using PPO clipping objective and $\simeq 150$ at iteration 140 for A2C using policy gradient one), the actor-critic algorithm using the PPO clipping objective achieved better result with the same configuration($\simeq 200$ at iteration 140).

If I had more time, I would have tried several epsilon values to find the optimal ones.

Here is the list of commands executed for this experiment:

rem: actor-critic curves are extracted from the results of the experiment 7.

```
# command 1
python run_hw3.py env_name=InvertedPendulum-v2 rl_alg=ac ep_len=1000\
estimate_advantage_args.discount=0.95 n_iter=100\
computation_graph_args.n_layers=2 computation_graph_args.size=64\
batch_size=5000 computation_graph_args.learning_rate=0.01\
exp_name=q7_10_10\
computation_graph_args.num_target_updates=10\
computation_graph_args.num_grad_steps_per_target_update=10\
train_args.ppo_clipping=True train_args.ppo_k_epochs=20 \
train_args.ppo_epsilon=0.2

# command 2
python run_hw3.py env_name=HalfCheetah-v2 rl_alg=ac ep_len=150\
estimate_advantage_args.discount=0.90 \
scalar_log_freq=1 n_iter=150 computation_graph_args.n_layers=2 \
computation_graph_args.size=32 batch_size=30000 \
train_batch_size=30000 eval_batch_size=1500 \
computation_graph_args.learning_rate=0.02 exp_name=q7_c_10_10 \
computation_graph_args.num_target_updates=10 \
computation_graph_args.num_grad_steps_per_target_update=10 \
train_args.ppo_clipping=True train_args.ppo_k_epochs=20 \
train_args.ppo_epsilon=0.2
```

Appendix (updated default configuration file)

```
env_name: 'CartPole-v0'
ep_len: 200
exp_name: 'todo'
n_iter: 1
mpc_horizon: 10
mpc_num_action_sequences: 1000
mpc_action_sampling_strategy: 'random'
cem_iterations: 4
cem_num_elites: 5
cem_alpha: 1
add_sl_noise: True
batch_size_initial: 5000
batch_size: 8000
train_batch_size: 512
eval_batch_size: 400
seed: 1
no_gpu: False
which_gpu: 1
video_log_freq: -1
scalar_log_freq: 1
save_params: False
rl_alg: 'todo'

computation_graph_args:
   learning_rate: 0.001
   n_layers: 2
   size: 128
   ensemble_size: 3
   num_grad_steps_per_target_update: 1
   num_target_updates: 1

estimate_advantage_args:
   discount: 0.95
   gae_lambda: None
   standardize_advantages: False
   reward_to_go: False
   nn_baseline: False

train_args:
   num_agent_train_steps_per_iter: 1
   num_critic_updates_per_agent_update: 1
   num_actor_updates_per_agent_update: 1
   discrete: False
   ob_dim:  0
   ac_dim: 0
   ppo_clipping: False
   ppo_k_epochs: 20
   ppo_epsilon: 0.2
```